PART AD-A254 747

53.301-298

(2)

N PAGE

Form Approved
OMB No. 0704-0188

1 hour per response, including the time for reviewing instructions, searching existing data sources, llection of information. Send comments regarding this burden estimate or any other aspect of this ington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Aug. 1992	Final - 5/1/87-3/31/92

4. TITLE AND SUBTITLE Transputer Implementation of the TRACE Model of Speech Recognition

5. FUNDING NUMBERS
C - DAAB07-87-C-H027

6. AUTHOR(S)
Jeffrey L. Elman/Paul S. Smith/Mark Dolson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The Regents of the University of California
University of California, San Diego
9500 Gilman Drive
La Jolla, California 92093-0526

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army
Communications-Electronics Command
Fort Monmouth, New Jersey 07703-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

DTIC ELECTE AUG 27 1992 S A D

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

We describe the accomplishments of the five-year project to develop an extended version of the TRACE model of speech recognition and to port it to a Transputer platform. We begin with an overview of the project. We then describe the TRACE Model. TRACE is an artificial neural network which attempts to model the processing of speech from acoustic input to word extraction, in a manner consistent with what is known of human perceptual abilities. Following that, we discuss the work done to extend and develop the model in the direction of real-time performance with an extended phonemic inventory. This includes algorithm changes to TRACE, software for managing the Transputer processors, a new acoustic/phonetic front-end, debugging and testing tools, and graphics visualization software.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Speech recognition; neural networks; parallel processing; transputers	23
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std Z39-18
298-102

53-85

Transputer Implementation of the TRACE Model of Speech Recognition: Final Report

Jeffrey L. Elman
Paul S. Smith
Mark Dolson

Center for Research in Language, University of California, San Diego

ABSTRACT

We describe the accomplishments of the five-year project to develop an extended version of the TRACE model of speech recognition and to port it to a Transputer platform. We begin with an overview of the project. We then describe the **TRACE** model. **TRACE** is an artificial neural network which attempts to model the processing of speech from acoustic input to word extraction, in a manner consistent with what is known of human perceptual abilities. Following that, we discuss the work done to extend and develop the model in the direction of real-time performance with an extended phonemic inventory. This includes algorithm changes to **TRACE**, software for managing the Transputer processors, a new acoustic/phonetic front-end, debugging and testing tools, and graphics visualization software.

I. PROJECT OVERVIEW

This report work done under contract to Army Avionics, Ft. Monmouth (contract #DAAB07-87-C-H027). The goal of the project involved development of a psychological plausible model of speech recognition which would operate in close to real-time. The model is based on a neural network architecture developed by Jeffrey Elman and James McClelland. By utilizing multiple sources of information in a parallel and interactive system, the model is robust in the face of a number of sources of variation, including ambient noise and coarticulatory effects.

The project had 4 major components:

1. Extension of front-end.

The initial version of TRACE had a limited front-end and was able to recognize only 15 phonemes. Most current phoneme-based systems rely on up to 60 phonemes; therefore, a necessary extension to TRACE was the development of a full-blown set of phoneme detectors.

This phase of the work involved a number of sub-components:

(a) recording, analysis, and marking of speech database. In order to control fully the parameters of interest to us, it was necessary to develop an in-house database which could be used for training and

testing of the front-end.

(b) Identification of the phoneme inventory. There is no single agreed-upon set of phonemes used in the industry. The closest to a standard is the so-called DARPAbet. This served as our initial candidate set, and was modified in response to our own testing.

(3) Acoustic pre-processing. Prior to being fed to the neural network, decisions needed to be made about how the digitized speech signal would be transformed (e.g., by filtering, log-scaling, normalization).

(4) Neural network phoneme processors. We experimented with a variety of neural network architectures for carrying out the phoneme recognition, including standard feed-forward networks, recurrent networks, and time-delay neural networks (TDNNs). We ultimately settled on a variant of the TDNN: the pairwise discriminant TDNN.

(5) Testing and evaluation. The phoneme estimation techniques provided by the front-end were tested extensively. Tests were carried out on (a) the in-house database, (b) the TIMIT database, and (c) a custom database provided by Army Avionic (and recorded in-house).

2. Transputer development software.

At the time the project was begun, there was very little development software available for the Transputer. In fact, the UCSD group ultimately played a significant role in the off-site development (by private vendors) of C and Sun-based software. Our group was one of the first major users of the Transputer and was thus able to influence and assist in the development of products.

1. Debugging. Although there is widespread recognition of the computational benefits of parallel processing, development tools are for the most part non-existent or in an immature state. In-house tools were therefore developed. These tools were used to analyze and debug processes running in a multi-processor environment, as well as to assess timing and performance.

2. Transputer operating system. A second effort was undertaken to develop software which would provide services typically offered by an operating system: control over process scheduling; memory management; I/O streams; error-detection.

3. CAC-QUAD testing. Once the QUAD boards were developed and made available, a third software project was initiated. This effort involved creation of software tools for testing and evaluating the QUAD boards provided by the Army.

3. Transputer-based TRACE implementation.

TRACE was initially developed without regard for how it might be implemented on a parallel platform. In order to port the system to a parallel and concurrent environment, fundamental changes were required in the software design. TRACE was essentially re-written from the ground up.

The transputer-based version of TRACE is optimized in order to take advantage of the hardware environment, and specifically, to utilize both the high-speed inter-processor serial communication links and the shared memory.

4. Graphics and visualization software.

A final software effort was directed at developing graphic tools in order to visualize the performance of the model. Given the concurrent activity of a very large number of (simulated) nodes in the

network, graphics visualization is a critical tool for localizing problems and analyzing performance. Tools were also created in order to visualize the speech data so that it could be marked and edited.

PERFORMANCE and RESULTS

Each of the sections below contains a detailed report of the results of the specific sub-components of the project. Software deliverables have separately been conveyed to Army Avionics at the time of the August, 1991 site visit.

As of this date, TRACE now exists as a functioning system. Equivalent implementations exist for both the Sun workstation (SPARC) and for the Army-based transputer boards.

TRACE has been trained to be a single-speaker system. Performance in single-word mode has not been measured quantitatively (because our goal has been continuous speech recognition) but is very good. TRACE also shows some robustness in the face of speech from users whose speech is novel.

As a phoneme recognizer, TRACE's performance is very good. It is difficult to compare performance with other systems, since recording conditions and target phonemes differ considerably across systems. However, it is our impression that TRACE's performance ranks near the top of any reported system.

In continuous mode, TRACE's performance on the Army command list is highly variable. Some utterances are recognized perfectly; other utterances are problematic and are only partially recognized. We have discovered that the current architecture is overly brittle in the face of differences in duration; thus, utterances which contain words whose duration differs significantly from that during training tend to derail the system. We have experimented with several solutions to this problem but are not satisfied it has been solved. Thus, dealing with durational differences remains a significant goal for any future development.

The remainder of this report describes the project in greater detail. Section II provides the theoretical motivation for the system. Sections III describes the transputer-specific software, and section IV describes the TRACE-specific software; this section is in turn broken down into a section on the acoustic/phonetic front-end and a section on the phoneme and word processing. Section V describes the graphics visualization software that was developed for the project.

II. TRACE MODEL

It is a curious paradox that some of the tasks which humans carry out with the least conscious awareness and with the greatest facility are precisely those tasks which seem to be the most complex and have been most resistant to machine-based implementation. Speech recognition and language understanding are particularly striking in this regard. Human listeners have little problem understanding language, even under circumstances which might be considered to be adverse. No machine-based recog-

[1] The treatment of time in TRACE is not entirely satisfactory The reduplication of connections is hardly elegant the many tokens (of features, phonemes, and words) are not bound together in any way by units which stand for types independent of their instantiation. Another, more satisfactory, approach to dealing with sequential inputs is described in Elman (1989; 1990).

nition system yet developed comes close to exhibiting an equivalent behavior.

The fact that the only physical system which is known to have solved the speech understanding problem is the human brain suggests that there may be aspects to brain-style computation which are particularly important for success in this task. This insight, of course, is what has prompted the recent revolution in neural computation, giving rise to what is variously called *connectionism*, *artificial neural network modeling*. The current project has involved applying a neural network architecture to the problem of speech recognition; this architecture is called the TRACE model.

Previous approaches

Most attempts at understanding how humans process and produce language have taken the standard digital computer as the metaphor for how the brain works. The metaphor is not always explicitly acknowledged, but is manifested in several important ways:

- It is often assumed that logical functions are confined to "boxes", which are connected by arrows indicating a restricted flow of information.

- Information processing is subject to a "digital flow of information", i.e., a problem-solving strategy consisting of *hypothesize, test,* and *branch*. Only a single hypothesis is considered at a given moment in time.

- Memory is conceived of as essentially passive. Information is stored in modules on a random basis. Efficient retrieval of information may be facilitated by some clever organizing scheme (e.g., alphabetic, frequency of use, semantic or phonological features) but ultimately the access of memory is decoupled from its contents.

All of these characteristics, it will be observed, are true of the von Neumann machine. They are characteristics which have led to demonstrable success in many areas of computation. What is wrong with such a model?

It is not at first obvious that the von Neumann/brain metaphor is in fact wrong; or, even if it is, that it matters much. The problem becomes apparent only when one takes the theorizing seriously enough to try to build a system on it. Then the failures are readily apparent. The clearest example of this comes from the field of speech recognition. Here a variety of approaches have been tried, and many of them (e.g., Hearsay, HWIM) have attempted to bring findings from linguistics and psychology to bear on the goal of getting computers to at least recognize (if not understand) human speech. Such attempts have, in general, been quite disappointing. In the end, the most successful systems—from the viewpoint of giving real-time performance in a limited but well-defined domain—have been those that eschew psychological or linguistic considerations. Rather than attempting to understand how it is that humans accomplish the problem of speech recognition, and replicating that process in silicon, these approaches adopt a brute-force stance.

While apparently expedient, this approach is not without its price. It leads to performance which is highly constrained. Such systems typically work well in limited environments, in which there are a small number of speakers, using a limited vocabulary with low-branching syntax, and in low noise. In the long-term, then, one can argue that such expediency leads to a poor solution.

The TRACE Model

The **TRACE** Model (Elman & McClelland, 1984; 1986; McClelland & Elman, 1986) was developed in an attempt to understand a number of specific problems in speech perception. We first describe that model, with additional extensions, and then in the next section report the results of selected simulations of the model.

Description. **TRACE** identifies three levels of organization: the *feature* level, the *phoneme* level, and the *word* level. Processing is carried out within the model over a large number of very simple (neuron-like) processing elements we call *nodes*. Nodes can be thought of as hypothesis detectors; there are feature nodes, phoneme nodes, and words nodes. Nodes have associated with them a numeric value that represents their *activation level*; this value changes over time and indicates the relative strength of the hypothesis represented by a given node.

Connections. Nodes are connected to one another in non-arbitrary ways which represent mutual compatibility or incompatibility of their hypotheses. Nodes whose concepts are mutually consistent (e.g., that the word "pan" contains the phoneme /p/) have excitatory connections. Those nodes whose concepts are inconsistent (e.g., that the phonemes /p/ and /a/ are simultaneously present in the input) have inhibitory connections. These connections, in addition to being either excitatory or inhibitory, are weighted with numeric values (multipliers) which indicate the extent of (in)compatibility.

Time: The TRACE. So far, what has been described is a network which allows us to represent words, phonemes, and features, as well as for the relationships between these different units. Time is an important additional variable in speech. The same sound may occur many times in an utterance, and it is important to bind the processing of that sound to the moment in time in which it occurs to preserve independence of the processes which are time-bound, while allowing for the interdependence which arises when one portion of an utterance sheds light on the identity of another.

We accomplish this by representing time as a series of networks, each identical and resembling what is sketched above, but each responsible for processing a given portion of the utterance These networks constitute an active memory framework there is no real distinction between short-term memory and the basic mechanisms of perception in which network is receiving the current input; as time progresses, input is directed toward successive networks called TRACEs, because they provide a TRACE of the recent past.

Processing. Processing begins with the TRACE networks quiescent. When speech comes in, it is directed toward the feature nodes in the first TRACE. These become activated to the extent that the features they stand for are found in the input. As time progresses, the input is directed toward later networks. (After some period of time, when the very early part of the utterance has been completely processed and its interpretation cannot change, the first networks are "re-used".)

In the original version of TRACE, feature nodes were arranged in groups of eight nodes; each group was responsible for detecting a certain feature, and nodes within a group had different ranges of sensitivity to the feature (from strongly present to strongly absent). In the newer version, this layer has been absorbed by a Time Delay Neural Network

(TDNN) which takes (filtered and parameterized) digitized speech as input, and yields phoneme estimates directly as its output. Feature processing is thus subsumed within the hidden layers of the TDNNs. The TDNN output then activates the phoneme nodes in the appropriate TRACEs.

Within a TRACE, phoneme nodes have inhibitory connections to each other; they also have excitatory connections to word nodes. Word nodes occupy the TRACEs corresponding to the time at which they begin. Phoneme nodes excite word nodes which contain them in that TRACE position. Thus, in a given TRACE, a /p/ node excites the word node for "pan" (which, since it begins in that TRACE, also is located in the same TRACE); it also excites the word node for "nap" (which is located in a prior TRACE, since the word itself must have begun earlier in time). Following the rule that inconsistent concepts compete with each other, word nodes in a TRACE are connected by inhibitory links.

This architecture has been tested extensively via computer simulations, and has been demonstrated to exhibit a number of properties which are characteristic of human speech processing. The model is highly robust in the face of noise and is able to use top-down (i.e., higher-level) knowledge information to compensate for degraded input. The model also treats coarticulatory variation as a source of information, rather than a meaningless corruption of the signal.

The early version of TRACE, however, was very slow (because it was simulated on serial hardware) and was implemented for only a subset of phonemes in English. In the current project, the algorithm has been ported to a parallel hardware environment, and has been extended to deal with the full range of phonemes in English.

In the remaining sections we describe the steps necessary to implement the model as a production system. Section III describes the software development which was oriented primarily toward the novel hardware environment that was developed for this project. Section IV describes the software effort specific to extending TRACE itself; this includes both the neural network acoustic/phonetic subsystem as well as the phoneme/word processors. Finally, Section V describes the graphics visualization software that was developed to support the project.

III. TRANSPUTER DEVELOPMENT SOFTWARE

A suite of development tools have been created for use on the transputer platform. These fall into several categories: (a) debugging, (b) compiler, (c) o/s services, (d) Unix simulator, and (e) various routines for testing the CAC-QUAD hardware.

TDB. Transputer Debugger.

This is the symbolic debugger which runs on the Sun 3, with the VME B011 card. It is currently being ported to the SPARC station. This has proven a very useful development tool for transputer applications.

Logical Systems C.

C cross compiler/assembler/linker/loader running on Sun and ibm PC platforms, generating code for transputer.

The cross compiler itself, including UCSD extensions to support multiplexed I/O, the B011 board (eventually the SPARC board as well).

O/S services.

• Minroute (transputer routing/ multiplexing code for I/O support). UCSD developed routing code. Very useful for debugging, and necessary for typical applications on multiple processors.

• Memory management.

• Remote Procedure Call services to Unix host.

• Interprocess and interprocessor communication.

Simtrace.

Stand alone simulator under Unix. This code is for predicting behavior of TRACE on the transputer platform and experimenting with algorithmic changes prior to implementing them on the CAC-QUAD.

CAC-QUAD hardware tests.

• T-800 processor test.

• On-chip memory test.

• Off-chip memory test.

• Shared memory test.

• Link communications test.

• Speed/performance test.

• Multi-processing test.

IV: TRANSPUTER IMPLEMENTATION OF TRACE

Currently TRACE is able to process speech consisting of up to 40 different phonemes, and theoretically an infinite number of words. As long as a suitably prepared lexicon exist, TRACE is able to be automatically reconfigured for any new lexicon. As the number of words in a lexicon increase, so will the resource requirements for TRACE. The most "taxed" resources will be available memory and available computational power. We commonly run TRACE on a lexicon of 100 to 200 words, and 40 phonemes.

The development of TRACE has been divided into three major subcomponents, and we discuss these separately below. In Section A we describe the process the digitized speech signal is converted into phoneme estimates. In this section we summarize the recording process, speech database, acoustic processing, and neural network phoneme processors. In Section B we describe the second major subcomponent of TRACE, which includes phoneme/word interactions and culminates in actual recognition of words. Finally, in Section C we describe the software interface with the CAC-QUAD hardware platform.

A. Front-end processing.

1. Recording, analysis, and marking of speech database.

The success of any speech recognition scheme is closely tied to the size, accuracy, and generality of the speech database upon which it is trained and tested. The in-house development of such databases has historically been an important component of the speech recognition research effort. Over the lifetime of the present project, three different databases have been employed:

• The speaker-dependent MBD1 database was developed at UCSD for the purpose of early testing and development. It contains 216 exemplars each of the stop consonants [b, d,, g, p, t, k] in consonant-vowel- consonant settings (e.g., "bib", "did", ...). The onset of each phoneme was marked by hand with the aid of special-purpose graphical analy-

sis software also developed at UCSD for this project.

• The speaker-independent TIMIT database was developed at Texas Instruments and MIT and became available midway through the project. It contains 10 sentences each for 630 speakers using a 5000 word vocabulary. Phoneme onsets have been hand marked for each of 62 different phonemes (yielding over 160,000 exemplars in all). This database was useful for development, but it had too little data for any single speaker to support speaker-dependent training.

• The speaker-dependent MBD2 database was developed at UCSD for the purpose of advanced testing and development. It contains 120 random, five-word sentences constructed from the 100-word Army Avionics vocabulary in Appendix A. Phoneme onsets were marked by hand (as in the MBD1 database) for each of 40 phonemes, resulting in 3126 separate exemplars. This is the database upon which all final results are based.

2. Identification of the phoneme inventory.

Based on the Army Avionics lexicon and on testing experiences, 40 phoneme labels were ultimately selected for use in this project. Of these, 37 correspond directly to phonemes in the standard "ARPABET", one label is for silence, one is for non-speech sounds, and a final label is expressly set aside for the "or" sound in words such as "four". A complete list of all 40 phoneme labels is given in Appendix B.

3. Acoustic preprocessing.

The speech is initially digitized at a 10 KHz sample rate. A number of transformations are then performed to convert the sampled speech into a form more appropriate for input into the phoneme-estimation neural network. These transformations are intended both to model (in a simplified manner) the response of the cochlea in the human ear, and to minimize any systematic variations in the recorded sound across different recording sessions. In addition, they serve to significantly reduce the amount of data presented to the neural network. Similar transformations are employed in a wide variety of contemporary neural-network-based speech recognition systems.

The TRACE speech preprocessing transforms the speech into a sequence of "frames" with a 10-msec interval between frames. Each frame ultimately consists of 16 parameters computed as follows:

• A 20-msec excerpt of the digitized speech is weighted by a Hamming window, and its spectrum is computed via a 256-point Fast Fourier Transform (FFT).

• The magnitude spectrum is collapsed into 15 logarithmically spaced frequency bands by combining adjacent channels of the FFT. These 15 values are then converted to dB, relative to the maximum value in that frame. (Hence, each frame has a 0 dB independent of all other frames.)

• A 16'th. parameter gives the total power across the entire spectrum for that frame. This value is converted to dB relative to a fixed reference.

This has the effect of replacing the 10,000 sample values originally associated with a single second of speech with 1,600 psychoacoustically- motivated parameter values. All parameters in all frames are then renormalized according to the following scheme:

• For each of the 16 parameters,

the mean and standard deviation of that parameter is computed across all available speech frames in a given recording session.

• Each parameter value is converted to a z-score by subtracting the mean value for that parameter and dividing by the standard deviation for that parameter.

This renormalization has the desirable effect of eliminating gross spectral differences between recording sessions (due, for example, to different microphones or microphone placements), but it has the UNDESIRable effect of requiring that such statistics be available in advance of any attempted speech recognition.

4. Neural network phoneme processors.

Time Delay Neural Networks

The phoneme estimation algorithm is based on the Time Delay Neural Network (TDNN) pioneered by A. Waibel at Carnegie-Mellon University. The basic idea is illustrated in Figure 1. The network consists of four layers of units, with the units in each layer organized into columns. Each unit computes a weighted sum of inputs, passes this number through a nonlinear function (the "logistic" function), and sends out the result as its output.

The first layer (the Input Layer) has 15 columns of 16 units each, corresponding to 15 successive frames of speech with 16 parameters in each frame. These 240 units provide excitation to the second layer (Hidden Layer 1) which has 13 columns of 8 units each. These 104 units, in turn, provide excitation to the third layer (Hidden Layer 2) which has 9 columns of 3 units each. Lastly, these 27 units provide excitation to the Output Layer, which (in this case) has a single column of 3 units, corresponding to three different phonemes.

The distinctive feature of this network is that two crucial constraints are placed on the connections from units in one layer to units in the layer above. The first constraint is that units receive excitation from only a fixed subset of the units in the layer below. For example each unit in the first column of Hidden Layer 1 receives input from every unit in the first *three* columns of the Input Layer (48 units in all). Each unit in the second column of Hidden Layer 1 receives input from every unit in columns two through four of the Input Layer (again, 48 units in all). This pattern continues for each column. Hence, each unit in the rightmost column in Hidden Layer 1 receives input from every unit in the three rightmost columns of the Input Layer (still 48 units in all).

This constraint is also applied to the connections from Hidden Layer 1 to Hidden Layer 2, except that each unit in the second hidden layer is allowed to collect excitation over *five* consecutive columns in the first hidden layer. The result of this arrangement is that each unit in Hidden Layer 1 receives exactly 48 weighted inputs from the Input Layer, while each unit in Hidden Layer 2 receives exactly 40 weighted inputs from Hidden Layer 1.

The second crucial constraint is that each unit in a given *row* is required to have the *same** set of weights as all other units in that row. This makes the units "time invariant." It means that if the pattern of excitations in the Input Layer is shifted one column to the left, then the pattern of excitations in Hidden Layer 1 (and, similarly, in Hidden Layer 2) will also be shifted one column to the left.

The output units are a special case. Each of the three output units receives input from all nine units in the corresponding row of Hidden Layer 2. Moreover, all nine units contribute equally (i.e., all nine weights are the same). The result is that the output units essentially serve to integrate the excitation across each of the corresponding rows of the second hidden layer.

To use this network as a phoneme detector, successive frames of input are presented to the rightmost column of the Input Layer. Because of the time invariance built into the network, each new frame has the effect of shifting all the excitations in the network one column to the left. Whenever a sufficient portion of the past 15 input frames contains a phoneme represented by one of the output units, that output unit will tend to be highly excited.

Obviously, the success of this network in actual applications depends entirely on the weights on the various connections between units. In total, there are 384 different weights between the Input Layer and Hidden Layer 1, and another 120 different weights connecting the two hidden Layers. These weights are all automatically and iteratively determined during a lengthy training phase via the back propagation learning algorithm. A cross-validation procedure is used to decide when learning is complete.

One other issue that arises with networks of this type is how to most efficiently train the network to discriminate not merely between three phonemes but between forty. It is possible to simply have forty output units and to train the network as above, but a more attractive approach is to first train separate networks to discriminate between specific subsets of phonemes (e.g., "b" vs. "d" vs. "g"), and then to train yet another network to discriminate between phonetic categories (e.g., voiced stops vs. unvoiced stops). The input-to-hidden-layer weights of these networks are then brought together as the front end of a hybrid network. These weights are kept fixed while the remaining weights in the hybrid network are iteratively adjusted to yield the best recognition performance for the entire set of phonemes. This requires considerably less training time for the same ultimate performance.

Pairwise-Discriminant Time Delay Neural Networks

In practice, the standard TDNN described above has proven vulnerable to two kinds of errors. First, the between-category discrimination task (e.g., stops vs. fricatives) is often sufficiently difficult that the hybrid approach proves impractical. Second, TDNN classifiers tend to make "hard" decisions as opposed to "soft" ones. This means that the TDNN network usually activates a single output strongly, independently of how certain the network is about the decision. Thus, there is no way to know from looking at the network outputs whether a second phoneme may have been nearly as good a match as the phoneme chosen by the network. The Pairwise-Discriminant TDNN (PD-TDNN) (originally proposed by Takami and Sagayama at ATR Laboratories in Japan) is a variant of the TDNN which avoids these two problems.

The PD-TDNN is a four-layer network whose input and hidden layers are identical to the TDNN described above. The difference between the two is that the PD-TDNN has a single output unit, and the activation function of this output unit is a modified logistic function: an ad-

ditional plateau is inserted so that an output of 0.5 is as stable as an output of 0.0 or 1.0. (Whereas the logistic function looks like a smoothed step function, the modified logistic function looks like a smoothed sequence of two steps.) Thus, the PD-TDNN output unit is essentially a tri-state logic unit.

With this modified output unit, the PD-TDNN can be trained according to the following rules:

- If the input contains phoneme #1 (say, "b"), the output is 0.0.
- If the input contains phoneme #2 (say, "d"), the output is 1.0.
- If the input contains neither phoneme #1 or #2, the output is 0.5.

The disadvantage of this approach is that a separate PD-TDNN is required for every phoneme pair that is to be discriminated. For example, one PD-TDNN discriminates between "b" and "d", a separate PD-TDNN discriminates between "b" and "g", yet another PD-TDNN discriminates between "b" and "p", and so on. The total aggregated output for each phoneme is determined by adding up the outputs of all PD-TDNN's whose target was 1.0 for that phoneme (and by adding [1.0 - the output] for all PD-TDNN's whose target was 0.0 for that phoneme).

On the other hand, the advantage of this approach is that the aggregated outputs tend to represent much "softer" decisions than the TDNN outputs. This can be seen in Figure 2 which compares the output of a TDNN to that of a group of PD-TDNN's in response to 25 presentations of the phoneme "d". The horizontal axis shows the output of the "d" unit, while the vertical axis shows the output of the most active output unit other than "d". Ideally, all the data points would lie to the right of the diagonal line (indicating that the "d" output is always more active than any other in response to a "d"). In practice, both the TDNN and the PD-TDNN's produce erroneous outputs for five of the 25 presentations. The important distinction, though, is that the erroneous outputs for the PD-TDNN's are still fairly far to the right on the graph (indicating a strong possibility that the correct answer is actually "d"). In contrast, the erroneous TDNN outputs are clustered near the vertical axis, suggesting that there is little chance of the correct answer being "d".

The PD-TDNN approach also allows resources to be focused explicitly on the discrimination tasks that are most demanding. By allocating networks in this fashion, we arrived at a set of 127 separate PD-TDNN's which jointly discriminate amongst 40 different phoneme labels.

5. Testing and evaluation.

Comparisons of different speech recognition algorithms are notoriously difficult because small differences in databases and testing paradigms can have a significant effect on the measured performance. In evaluating phoneme estimation algorithms, an additional issue is that their ultimate utility depends solely on the extent to which they facilitate accurate word recognition in an integrated system. Nevertheless, tests and comparisons are an indispensable part of the development process. Our most significant findings can be summarized as follows.

Eight separate TDNN sub-nets were ultimately trained and evaluated on the TIMIT speaker-independent database. Each sub-net was specialized for a different category of phoneme (e.g., voiced stops, unvoiced stops, voiced fricatives, etc.), and phoneme-identification accuracy ranged from 60% for the vowels

to 84% for the stops. These results were comparable to those obtained by other research groups for this database with similar approaches. However, the problem with this approach was that TDNN's trained to discriminate *between* the various categories achieved accuracies ranging from only 52% to 84%. This led to the development of the in-house, speaker-dependent MBD2 database and to the rejection of the multiple sub-net approach.

To establish a baseline performance on the MBD2 database, a monolithic TDNN was trained to identify all 40 phonemes. This network achieved an accuracy of 52% under realistic test conditions (phonemes presented in all possible time-alignments relative to the TDNN). However, as discussed above, it tended to strongly activate a single output rather than partially activating several. In contrast, a collection of 127 PD-TDNN's achieved an accuracy of only 54% under the same testing conditions. More significantly, though, the PD-TDNN identified the correct phoneme as one of its top three choices 80% of the time whereas the TDNN included the correct phoneme within its three most active outputs only 68% of the time. It is likely that the PD-TDNN performance can still be significantly improved simply by increasing the size of the MBD2 database, but the amount of this improvement remains to be determined.

B. Phoneme/Word Interactions

The result of the front-end processing just described is a set of phoneme estimates for each cycle. These are fed directly to the layer of phoneme processors, which are in turn connected to word processors.

Phoneme layer. Phonemes receive estimates from the PDNN layer reflecting input to the current cycle. Phonemes also receive excitatory input from words in previous, future, and the current cycle. Phonemes receive inhibitory input from other phonemes active in the current cycle.

Phonemes provide inhibitory output to other phonemes in the current cycle as well as excitatory and inhibitory output to words which begin in the current cycle, and to words which might be active during this cycle (words which began in a previous cycle, but span into the current one).

Phonemes are all modeled as being 20ms, or one TRACE cycle, in duration. Because the actual duration of a given phoneme is almost certainly much longer than this, the same phoneme will be reactivated for several consecutive cycles.

Word Layer. Words receive excitatory and inhibitory input from active phonemes in any cycle for which the word exist or spans, and inhibitory input from other words which are coexistent in any cycle.

Words are modeled as being constructed of variable length phonemes. Each word in the lexicon is described by a template which was developed based on empirical study of digitized speech. The templates consist of a minimum onset time for each phoneme in the word, and a maximum offset time. For example, the word "seven" is represented by the phonemes 's' 'E' 'v' 'E' 'n' in sequence. The actual lexicon entry for this word looks like this:

sEvEn:

s	0.00	0.19
E	0.11	0.30
v	0.21	0.33

E 0.27 0.41
n 0.30 0.78

The numbers next to each phoneme represent the earliest onset time in seconds of the phoneme, and the latest offset (end time) in seconds for the phoneme. For example the earliest we expect to see the first 'E' phoneme is about 0.11 seconds after we detected the start of the word, and the longest we expect it to last is 0.30 - 0.11 = 0.19 seconds. We see that the 'v' phoneme is allowed to start at 0.21 seconds after we detected the start of the word, this is in the middle of the allowable time frame for the previous 'E' sound. This is how TRACE accommodates variable rates. In effect TRACE allows all possible durations for all the phonemes in a given word. Of course the word representations are based on a limited, though large, data set so it is certainly possibl for some error from extremely slow, or extremely rapid speech.

The duration problem. Earlier version of TRACE represented the phonemes and words in a different way. Previously phonemes were represented as having a specific duration, based on empirical study of phonemes in speech. Words' representations didn't include any phoneme length information, only which phonemes were present in a given word. Experimentation with this representation exposed some problems in recognizing the same word spoken at two slightly different speeds, as is encountered in normal speech, even with the same speaker. The degree of variability in the duration of words, and indeed the phonemes within the words, proved to be too difficult for the fixed length word/ phoneme representations.

With the variable length phonemes (variable within each word that is) the algorithm was expected, and has proven to be much better at dealing with variable rates of speech. With the current representation of words and phonemes TRACE is perfectly happy to allow the same word or phoneme to be of a different length each time it is seen in the input stream. This has considerably increased the performance of TRACE in terms of correct recognition. Unfortunately this change in representation has led to about a 100 fold increase in the processing requirements for the algorithm. Some part of this increase has been reduced by various coding methods, but the time to process speech has been considerably increased.

C. Hardware platform.

There were several desiderata which were taken into account in choosing a suitable hardware platform for implementing the model:

• *processor speed.* The calculation of node activations involves floating-point multiplies and adds, and so it was deemed important to select a processor with high-performance arithmetic capabilities.

• *support for parallelism.* It was anticipated that the greatest improvement in performance would come from parallel operation. Therefore, we wanted a processor which (a) simulated parallel processing on-chip; and (b) was capable of functioning in a parallel mode with many other processors.

• *communications support.* Because the target architecture would involve a large number of processors (each supporting a large number of independent processes), we realized that it would be important to maximize communica-

tion bandwidth. Furthermore, we wanted flexibility in how this bandwidth was to be achieved. We envisioned a system in which communication bandwidth might occur either directly (e.g., along a bus or point-to- point links) or indirectly, via shared memory.

• *software compatibility with host.* TRACE had been developed in C code on conventional machines, and we anticipated continuing to develop the software in simulation form. We wanted the simulation software to be easily portable to the target machine, and back again for debugging purposes. It was thus important that the target processors be easily integrated into a network in which the simulation host (i.e., a VAX or Sun computer) might participate in a transparent fashion.

These considerations led us to chose the Inmos T-800 Transputer. The T-800 processor is a 32-bit CMOS microcomputer. It has an integral 64-bit floating point unit with 1.5 MFLOPS performance (at 10 MIPS). It supports 4K on-chip local memory, and can directly address 4GB total memory. There are four serial links (running at selectable speeds of 5, 10, or 20 Mbits/sec) which provide for inter-processor communication. The T-800 also has a hardware scheduler which provides for programming on-chip. There are implementations of several high-level languages, including C.

The current hardware configuration is organized around what we call Quad boards. In the current configuration, there are four Quad boards, plus a fifth IO board.

Each Quad contains four T-800 transputers and 1MB of shared memory. Each T-800 also has 512K of local memory. Transputers within a Quad communicate via shared memory. Finally, each transputer on a given Quad has serial link connections to one transputer on each of the other three Quads (leaving one serial link free).

The IO board has a single T-800, A/D chip, 512K local memory, and 512K of shared memory, accessible by Quad0. The T-800 on the IO board uses one of its four serial links to communicate with external devices (and for downloading code). This T-800 also collects digitized speech from the A/D, carries out feature extraction, and places the result in shared memory. The shared memory (with Quad0) is used both for booting the remaining quads and for transferring speech data to other processors.

Shared memory on each board can be controlled in one of two modes. In transparent mode, whichever transputer requests memory first gains access for the duration of a single I/O operation. Other requests during this period are deferred (transputers are placed in wait states until memory becomes available). When the I/O operation is complete, control of the shared memory is automatically shifted to the next transputer. In controlled mode, a transputer may lock the shared memory; when this happens, the controlling transputer blocks access from other transputers until it explicitly yields control.

Figure 3 shows a simplified schematic drawing of the system.

Functionally, TRACE is implemented in the following way. The IO board is responsible for the interface with the world. This includes downloading code, controlling A/D conversion, and communicating the results of TRACE's processing to an external host/device. The T-800 on the IO board is responsible for running a monitor program which

will supervise TRACE and detect faults and errors. It also reports TRACE results to the host/external world. This T-800 will also carry out initial acoustic/phonetic processing on the speech, and place the results in global memory which is accessible to Quad0.

Each of the Quad boards will run the same code, but will be responsible for a subset of phoneme and word nodes. Figure 4 shows the processes that will run on each of the four T-800s on any given Quad (the transputer here is labelled TP0).

In this example, TP0 is responsible for processing a subset of phonemes (shown here as k,l,m,...) and words (ending here in must). Data structures for the entire set of phonemes and words are maintained in shared memory (shown on the left as a,b,c,...,zulu). Inputs for the current TRACE are stored in *cur_excitation*; these are sent to the TRACE code on each transputer, which then carries out the node interactions and updates. The resulting new node excitations are placed back in shared memory in the *new_excitation* structure.

This design was motivated by the desire to balance computational issues with communication requirements. In the extreme, the computational performance would be maximized by allocating one node per transputer. This is the most natural mapping for a neural network, and is similar in spirit to the approach taken in the Connection Machine.

There are two problems with this. The cost, of course, would be excessive; and to some extent, it represents overkill. Each T-800 offers about 10 MIPS performance, but the computational processing carried out by each node is relatively simple; nodes are essentially integrators. The second problem is that this approach incurs an enormous communications overhead. Depending on the type of node, there may be anywhere from 100 to 1,000 node interconnects. Neither the transputer nor any silicon-based technology currently available allows this bandwidth.

These considerations led us to develop a hybrid architecture. By placing several transputers together on a board, and allowing for high-bandwidth communication on a Quad via shared memory, we in effect create a super-transputer, with 16 communications links and an aggregate of 40 MIPS performance. Each transputer on a Quad queries the shared memory for nodes-to-be-processed and removes that node from the pending queue (following the Linda scheme; Ahuja, Carriero, & Gelernter, 1986). The transputer then calculates the effect that this nodes will have on each of the other nodes in the entire network. The result is stored temporarily in the transputer's local memory. At the conclusion of the current time slice, information stored in the separate local memories is then integrated into the Quad's shared memory, and finally, with the information on other Quads. This scheme, we believe, makes it possible to achieve a balanced load while at the same time minimizing communication requirements over the serial links.

V. Graphics visualization software.

Because trace can produce a tremendous amount of useful information about the state of the network, and all of it's nodes, every 20ms it was necessary to devise some tools for interpreting this in-

formation. In order for a human to interpret an asses the output from trace it is necessary to analyze an output for each node (phoneme or word) on every cycle, and in turn analysis the same output for multiple cycles. If the network contains 250 nodes, and looks at 2 seconds of speech (100 cycles) that would require the analysis of 25000 (250*100) data points, just to determine the state of the network (i.e. what has been recognized) at the end of the current cycle. This analysis would have to be performed on every cycle up to the 100th (i.e. 2 seconds), which would ultimately mean looking at 2.5 million data points, just to determine the result after 2 seconds of speech have been seen.

Obviously analysing 2.5 million numbers in 2 seconds would not be a possible task for a human. We've develop a visualization tool which runs under X windows on a unix work station. This tool presents a graphical picture of of the state of the network for any given cycle, and is able to record and replay the state of the network for a large number of cycles. We have coupled this tool with the trace algorithm on the Unix environment to produce a program called xtrace. Xtrace allows a user to interactively, graphically, run trace. The user is able to alter various "tunable" parameters of the network on the fly via a mouse, or keyboard inputs. The tool allows the user to filter or combine information from all the nodes, at all times in the network, and display a comprehensible picture of the state of the network at any given time. This "state" very clearly shows which words TRACE has recognized at a given time.

Xtrace uses a tremendous amount of memory and disk space and so is not a suitable tool to be run on the transputer network, but it has been essential to the development of TRACE. While developing and experimenting with the algorithm it has been necessary to continuously adjust TRACE's parameters (weights for node connections, decay times, etc), in order to find a sensible set of parameters to allow the recognizer to correctly operate.

Some work as been put toward developing a "winner" algorithm. This algorithm would automatically analyze all the node output on every cycle and notify the user when it believe a word has been recognized. Various means of doing this have been experimented with, and we have met with some success, but this aspect hasn't been fully integrated with TRACE.

APPENDIX A: Small lexicon-1

[The alphabet: alpha, bravo, charlie, ..., zulu]

[The numbers: zero through twenty]

Thirty, forty, fifty, sixty, seventy, eighty, ninty, hundred, thousand

Point, first, second, third, last

O. & I.
A. & L.
S.2.
S.3.
air S.3.
T.O.C.
F.A.C.
N.C.S.
F.C.C.
F.S.O.
X.O.
F.S.E.
A.V.I.M.
J.A.A.T.
tac C.P.

disengage
execute
vicon
frequency
channel
command
divarty
base
medevac
tower
scout
niner
commander
support
brigade
battalion
company
clearance
authenticate

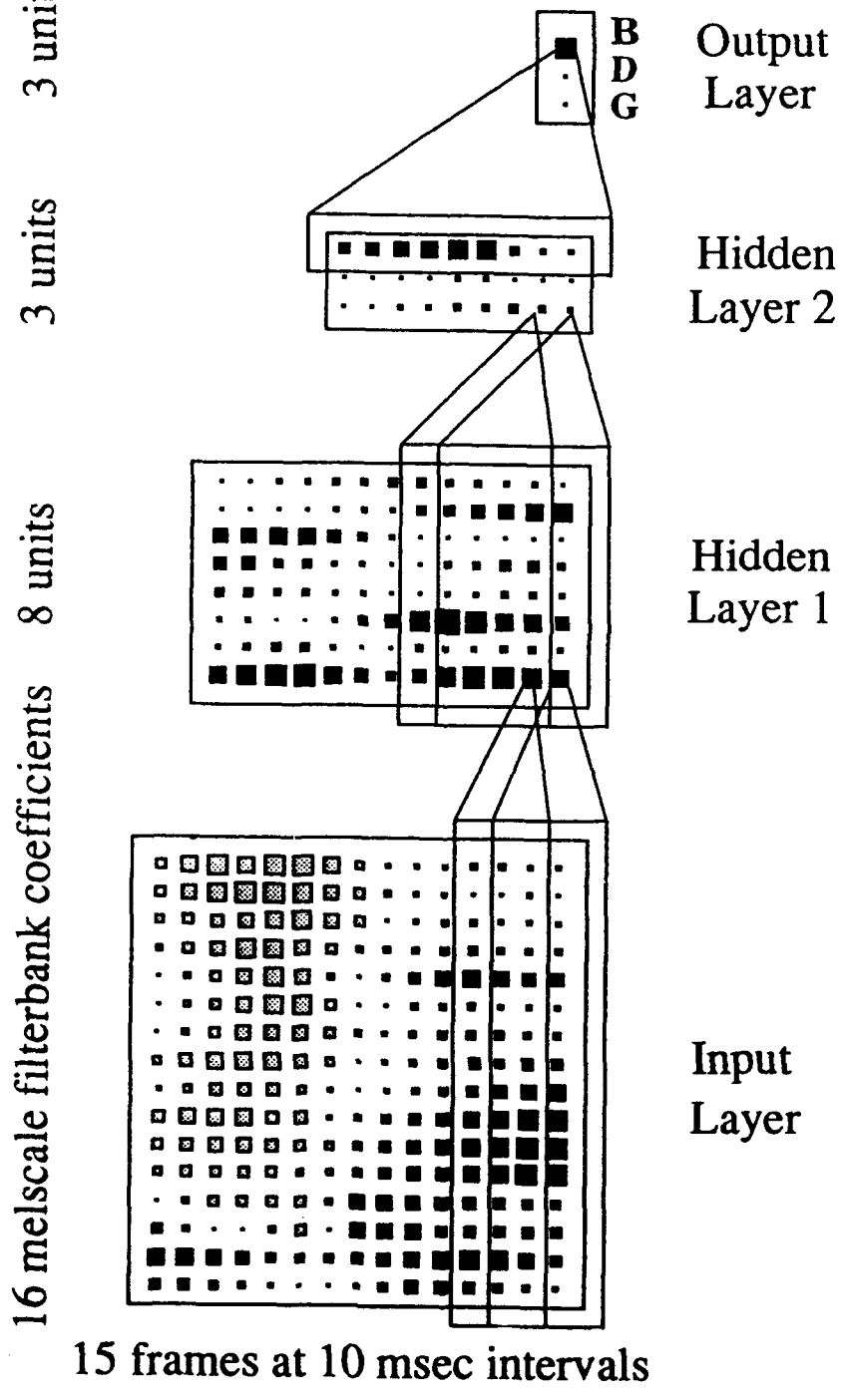
say again
give me
call sign
call up
look up
ground control
platoon leader

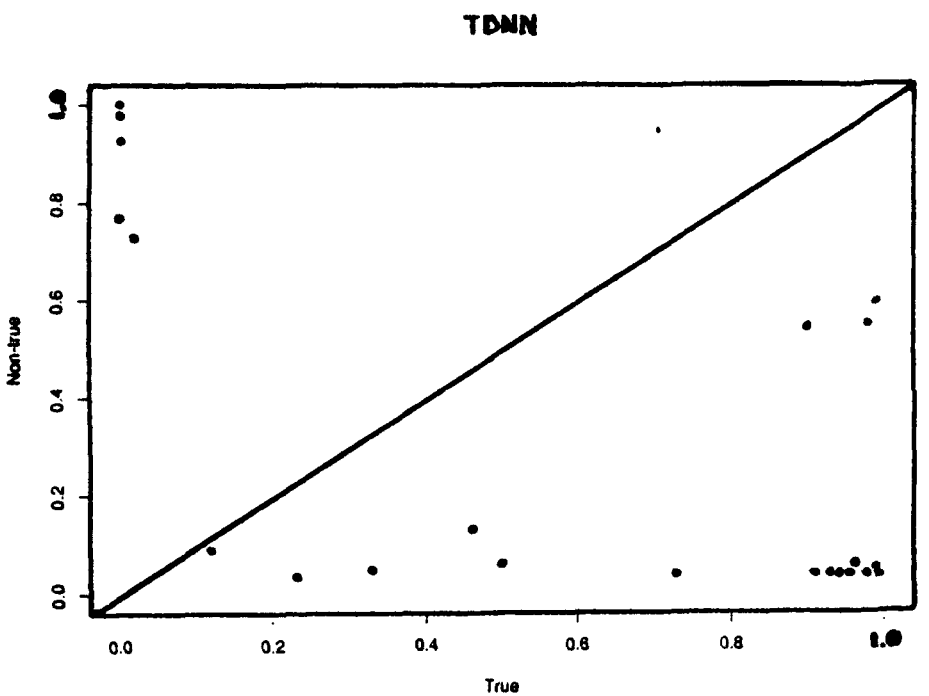
APPENDIX B: Phoneme Inventory

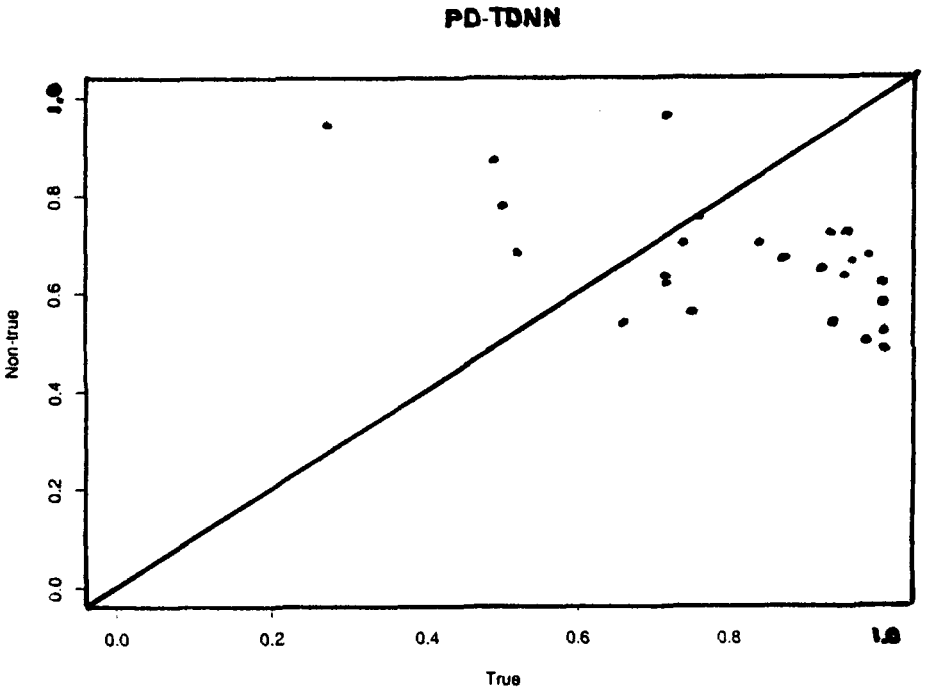
ARPAbet symbol	*TRACE symbol*	*Example in word context*
b	b	Bob
d	d	Dad
g	g	Gag
p	p	Pop
t	t	Tot
k	k	Kick
dx	F	baTTer
v	v	Very
z	z	Zoo
jh	J	Judge
f	f	Fief
s	s	Sis
th	T	THief
ch	C	CHurch
l	l	Led
r	r	Red
w	w	Wet
y	y	Yet
n	n	Non
m	m	Mom
hh	h	Hay
iy	i	bEEt
ih	I	bIt
eh	E	bEt
ae	@	bAt
ah	A	bUtt
ax	x	thE
aa	a	cOt
uw	u	bOOt
uh	U	bOOk
er	R	bIRd
or	4	fOR
el	L	bottLE

ey	e	bAIt
ay	Y	bIte
ow	o	bOAt
oy	O	bOy
aw	W	abOUt
sil	-	<silence>
?	?	<other>

TDNN Architecture




TDNN
Non-true
True
0.0
0.2
0.4
0.6
0.8
1.0


PD-TDNN
Non-true
True
0.0
0.2
0.4
0.6
0.8

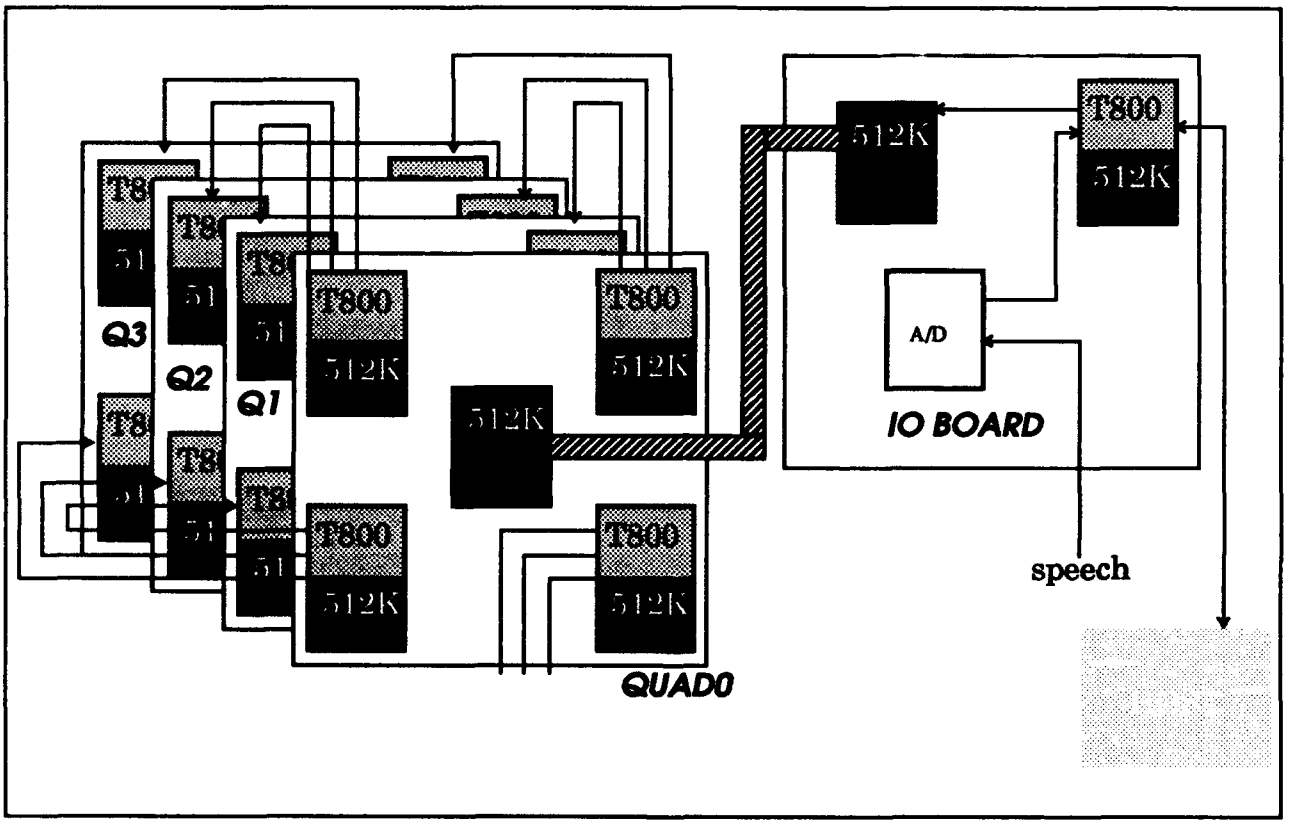


Figure 3

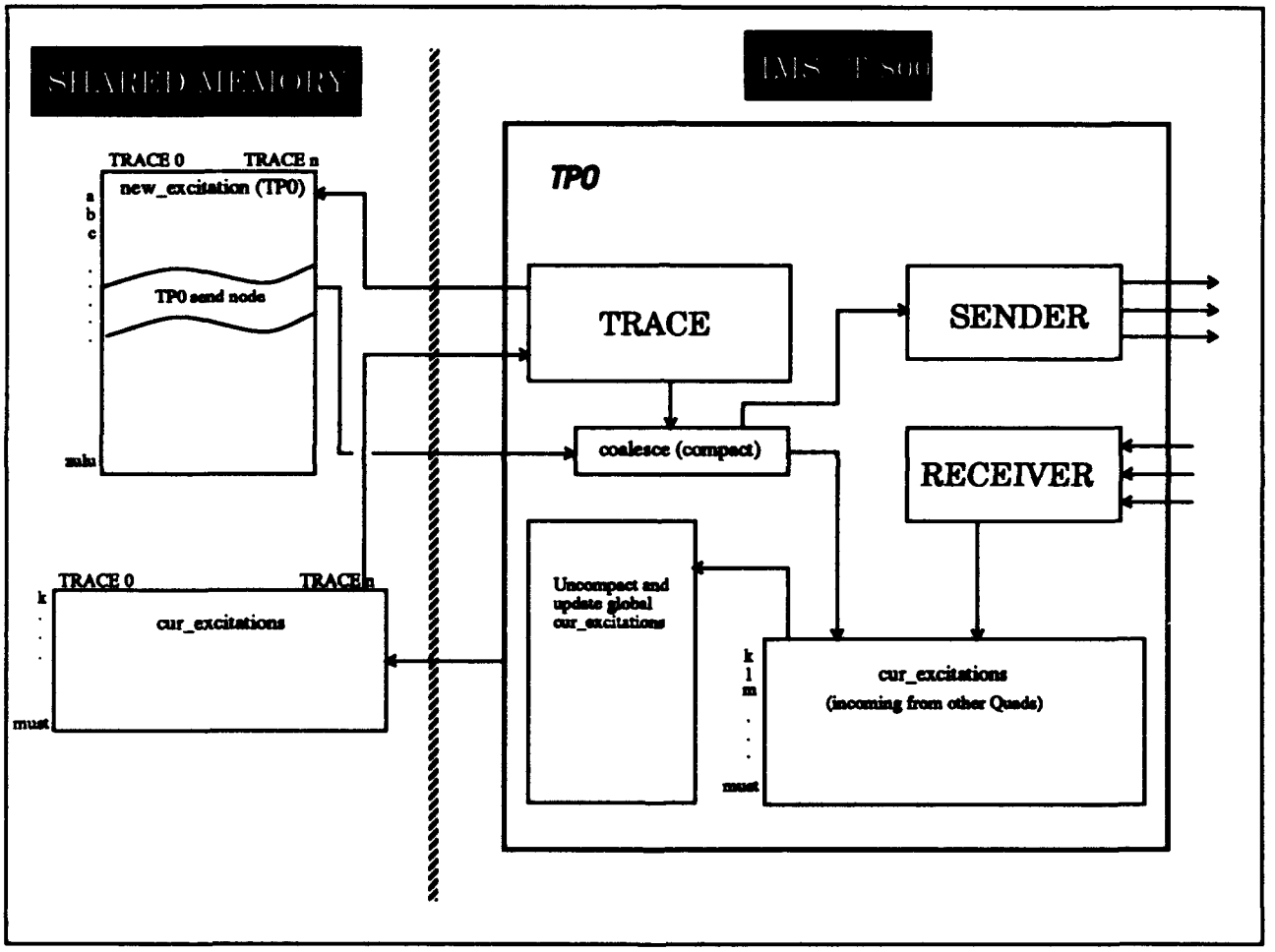

SHARED MEMORY
TRACE 0
TRACE n
new_excitation (TP0)
a
b
c
TP0 send node
nslu
TP0
TRACE
SENDER
coalesce (compact)
RECEIVER
TRACE 0
TRACE n
k
cur_excitations
must
Uncompact and update global cur_excitations
k
l
m
cur_excitations
(incoming from other Quads)
must

Figure 4